PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 12/00	A2	(11) International Publication Number: WO 99/26377 (43) International Publication Date: 27 May 1999 (27.05.99)

(21) International Application Number: PCT/US98/24395

(22) International Filing Date: 16 November 1998 (16.11.98)

(30) Priority Data:
08/972,157 17 November 1997 (17.11.97) US

(71) Applicant: MCMZ TECHNOLOGY INNOVATIONS LLC [US/US]; 11029 Grassy Knoll Terrace, Germantown, MD 20876 (US).

(72) Inventor: SCHUG, Klaus, H.; 3360 Gunnison Drive, Fort Collins, CO 80526-2790 (US).

(74) Agents: ROBERTS, Jon, L. et al.; Roberts & Brownell, LLC, Suite 212, 8381 Old Courthouse Road, Vienna, VA 22182 (US).

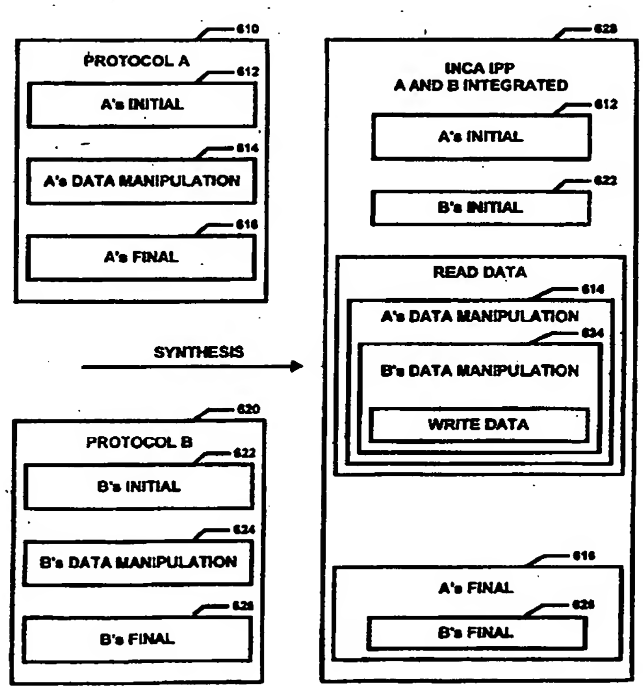
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: A HIGH PERFORMANCE INTEROPERABLE NETWORK COMMUNICATIONS ARCHITECTURE (INCA)

(57) Abstract

An interoperable, software only network communications architecture (INCA) is presented that improves the internal throughput of network communicated data of workstation and PC class computers at the user level, application program level, by 260 % to 760 %. The architecture is unique because it is interoperable with all existing programs, computers and networks requiring minimal effort to set up and use. INCA operates by mapping network data between the application and operating address space without copying the data, integrating all protocol execution into a single processing loop in the application address space, performing protocol checksumming on a machine word size of data within the protocol execution loop, and providing an application program interface very similar to existing application program interfaces. The network interface driver functions are altered to set up network data transfers to and from the application address space without copying of the data to the OS address space, while buffer management, application to message multiplexing/demultiplexing and security functions are also being performed by the modified network interface driver software. Protocols are executed in the application address space in a single integrated protocol processing loop that interfaces directly to the INCA NI driver on one end and to the application on the other end in order to minimize the amount of times that network communicated data must travel across the internal memory bus. A familiar looking application program interface is provided that differs only slightly from existing application program interfaces which allows existing applications to use the new software with a minimum of effort and cost.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

A HIGH PERFORMANCE INTEROPERABLE NETWORK COMMUNICATIONS ARCHITECTURE (INCA)

FIELD OF THE INVENTION

This invention relates generally to computer network communications. More particularly, the present invention relates to a method to improve the internal computer throughput rate of network communicated data.

BACKGROUND OF THE INVENTION

Network technology has advanced in the last few years from transmitting data at 10 million bit per second (Mbps) to near 1 Gigabit per second (Gbps). At the same time, Central Processing Unit (CPU) technology inside computers has advanced from a clock rate of 10 Million cycles (Hertz) per second (MHz) to 500 MHz. Despite the 500% to 1000% increase in network and CPU capabilities, the execution rate of programs that receive data over a network has only increased by a mere 100%, to a rate of approximately 2 Mbps. In addition, the internal computer delays associated with processing network communicated data have decreased only marginally despite orders of magnitude increase in network and CPU capabilities. Somewhere between the network interface (NI) and the CPU, the internal hardware and software architecture of computers is severely restricting data rates at the application program level and thereby negating network and CPU technology advances for network communication. As a result, very few network communication benefits have resulted from the faster network and CPU technologies.

Present research and prototype systems aimed at increasing internal computer throughput and reducing internal processing delay of network communicated data have all done so without increasing application level data throughput, or at the expense of interoperability, or both. For

purposes of this specification, network communicated data includes all matter that is communicated over a network. Present research solutions and custom system implementations increase data throughput between the NI of the computer and the network. However, the data throughput of the application programs is either not increased, or is only increased by requiring new or highly modified versions of several or all of the following: application programs, computer operating systems (OSs), internal machine architectures, communications protocols and NIs. In short, interoperability with all existing computer systems, programs and networks is lost.

A range of problems associated with network operations is still present. The present state of the art of increasing the performance of internal computer architectures:

1. Prevents computers from utilizing the tremendous advances in network and CPU technologies;
2. Fails to solve the problem by focusing mainly on NI to network transfer rate increases rather than on increasing network communicated data throughput at the application program level;
3. Severely restricts computer network communicated data throughput at the application program level to a fraction of existing low speed network and CPU capabilities;
4. Prevents the use of available and the implementation of new computer applications that require high network communicated data throughput at the application program interface or low internal computer processing delays;
5. Requires a massive reinvestment by the computer user community in new machines, software programs and network technologies because of a lack of interoperability with existing computer systems and components.

A need therefore exists for higher network communicated data throughput inside

computers to allow high data rate and low processing delay network communication while maintaining interoperability with existing application programs, computers, OSs, NIs, networks and communication protocols.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide high network communicated data throughput and low network communicated data processing delay inside computers.

It is a further object of the present invention to maintain interoperability with all existing computer programs, computers, OSs, NIs, networks and communication protocols.

It is a further object of the present invention to be useable on all existing personal computer (PC) and workstation (WS) class computers, as well as on most other, greater capability computer systems with only minor software modifications to the existing application programs and/or OS.

It is a further object of the present invention to dramatically increase the network communicated data throughput at the application level (not just NI to network).

It is a further object of the present invention to dramatically increase the network communicated data throughput at the application level (not just NI to network level) for small messages (less than or equal 200 bytes) and for large messages.

It is a further object of the present invention to speed up communication protocol processing for all levels and types of protocols.

It is a further object of the present invention to reduce the amount of times network communicated data is sent across the internal computer memory bus.

It is a further object of the present invention to increase the performance of non networking applications on networked computers by processing network management messages and messages not addressed to the machine at least four times faster than presently processed.

It is a further object of the present invention to be interoperable with existing computer

systems and network components and to not require costly changes to these components to improve performance.

It is a further object of the present invention to require only minor software changes to application program interfaces (APIs) or NI drivers.

It is a further object of the present invention to be installed, tested and operational in a short amount of time, i.e., minutes to hours.

It is a further object of the present invention to provide for application (not just NI to and from the network) throughput of network communicated data at high speed network transmission rates.

It is a further object of the present invention to enable the use and implementation of application programs that require high application level network communicated data throughput and low internal computer network communicated data handling delay.

It is a further object of the present invention to allow the utilization of high speed network and CPU technologies by enabling applications to process data at high speed network rates.

The present invention is a library of programs comprising three main programs integrated into one software library: a computer NI driver, an integrated protocol processing (IPP) loop and an API. The INCA NI driver comprises software that controls the NI hardware and transfers the network messages and data in the messages from the network to the computer's memory. The IPP loop software performs communication protocol processing functions such as error handling, addressing, reassembly and data extraction from the network message. The API software passes the network communicated data sent via the network to the application program that needs the network communicated data. The same process works in reverse for transmitting network communicated data from the application over the network to a remote computer. The existing computer components, e.g., the NI, CPU, main memory, direct memory access (DMA) and OS, are used with control or execution of these resources being altered by the INCA software

functions.

The advantage of using the present invention is that the existing, inefficient network communicated data handling is greatly enhanced without requiring any additional hardware or major software modifications to the computer system. INCA speeds up internal network communicated data handling to such a degree that data can be transferred to and from the application programs, via the network interface, at speeds approaching that of high speed network communicated data transmission rates. The CPU is required to operate more frequently on more data, utilizing the increased CPU capabilities. The present invention greatly reduces the number of times network communicated data must be transferred across the internal computer memory bus and greatly speeds up the processing of communication protocols, with particular improvement in the checksumming function.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows an overview of a typical existing network communication system.

Figure 2 shows an overview of the INCA network communication system.

Figure 3 shows an overview of the endpoint mechanism.

Figure 4a shows examples of the typical non INCA, non IPP for-loops used for protocol processing.

Figure 4b shows an example of a single, integrated INCA IPP for-loop used for protocol processing.

Figure 5 shows the INCA IPP stages of protocol execution.

Figure 6 shows the INCA IPP method of integrating various protocols into a single execution loop.

Figure 7 shows the alternative "system calls" comprising INCA's API.

Figure 8 shows INCA's performance improvement on WS class computers.

Figure 9 shows INCA's small message size performance improvement on PC class computers.

Figure 10 shows INCA's performance improvement with all standard message sizes on PC class computers.

Figure 11 shows INCA's management and control flow.

DETAILED DESCRIPTION OF THE INVENTION

Referring to **Figure 1**, an overview of a typical existing network communication system inside a computer is shown. When a network message **102** is received by the network interface (NI) **104**, the NI **104** sends an interrupt signal to the operating system (OS) **106**. The network communicated data is then copied from the NI **104** into the OS message buffers **110** which are located in the OS memory address space **108**. Once the network communicated data is available in the OS message buffers **110**, the OS **106** typically copies the network communicated data into the Internet Protocol (IP) address space **112**. Once IP processing is completed, the network communicated data is copied to the User Datagram Protocol (UDP)/Transport Control Protocol (TCP) address space **114**. Once the UDP processing is completed, if any additional protocols are used external to the application program, the network communicated data is copied to the Other Protocol address space **116** where the network communicated data is processed further. The network communicated data is then copied to the application program interface (API) address space **118**. Finally, the application reads the network communicated data which requires another copy of the network communicated data into the application program memory address space **120**. As illustrated in **Figure 1**, copying of the network communicated data occurs numerous times in the normal course of present operations.

Referring to **Figure 2**, an overview of the INCA network communication system is shown. Contrasting INCA to the typical network communication system in **Figure 1**, it is evident INCA eliminates several data copying steps and as a result, INCA performs in a more efficient manner for increasing a network's data throughput rate. In addition, INCA implements several other efficiency improving steps which will be discussed later.

As shown in **Figure 2**, the present invention comprises three main software components: an INCA NI driver **202**, an INCA IPP (execution loop) **204** and an INCA API **206**. These components reside inside computer message buffers **208** together with the current computer software components such as application programs **210**, operating systems (OS) **212**, communication protocols, and the current computer hardware components such as the NI **214**, system memory bus **216** and **218**, OS memory address space **220** and application program memory address space **224** and one or more CPUs, disks, etc.

The first component, the INCA NI driver **202**, is a software set of programming language functions that supports direct access to the NI **214** and performs the following functions:

1. Controls the NI device **214** and other involved devices (e.g., DMA) to set up a transfer of network messages **222** from the NI **214** to OS memory address space **220**;
2. Manages the NI **214** to computer memory transfer;
3. Demultiplexes incoming network messages to the proper recipient (i.e., application);
4. Provides protection of network communicated data from different applications;
5. Transfers the network communicated data to the application program memory address space **224**;
6. Interfaces to the INCA IPP **204** to control protocol execution;
7. Interfaces to the INCA API **206** to control application program network access;
8. Relinquishes any control over the NI **214** upon completion of message handling.

These eight functions are performed by the INCA NI driver **202** component for reception of network messages. In the case of the transmission of network communicated data from one computer to another computer over a network, the eight functions are performed in reverse order. Since the transmitting case is the same as the receiving case, it is not discussed separately. The following description of the INCA NI driver functions is for the receiving case. The INCA NI

driver component may include software linked to the INCA software library which is not typically considered NI device driver code.

The first two functions, control and management of transferring network communicated data from the NI device to internal computer memory (i.e., random access memory - RAM), or some other type of memory (e.g. cache, hard disk) are initiated when a message arrives at the computer from a network connection. The NI hardware signals the arrival of a message, typically a hardware interrupt. The message arrival notification signal is received by the INCA NI driver **202**. Upon receipt of a message arrival notification, the INCA NI driver **202** takes over control of the NI device (e.g., Asynchronous Transfer Mode (ATM) network card), and sets up the registers, firmware, etc., of the device to transfer the message from the device to main memory. Transferring the message or network communicated data is in response to the call functions of either an application program interface, an application program, a network interface device or a network interface driver.

The transfer can be accomplished via two main methods, via DMA or programmed input/output (PIO). In the case of DMA transfers of network communicated data between the NI **214** and OS memory address space **220**, the INCA NI driver **202** sets up memory and NI device addresses, message buffer sizes/alignments/addresses, and signals the start and completion of every transfer. If an error occurs, the INCA NI driver **202** attempts to resolve the error through such actions as reinitializing a DMA transfer or repeating transfers. At the completion of DMA transfers, the INCA NI driver **202** releases control of any DMA and NI **214**, releases any allocated message buffer memory **208**, and ceases execution.

In the case of PIO, the CPU transfers every byte of network communicated data from the NI to computer memory. The INCA NI driver **202** provides the necessary parameters, memory and NI addresses, transfer sizes and buffer sizes/alignments/addresses for the CPU to transfer the network communicated data to computer memory.

In the preferred embodiment, the OS **212** manages the address mapping between the virtual addresses of message buffers specified by an application and the physical addresses required for actual transmission and reception. In an alternative embodiment, the application program manages the address mapping. In yet another embodiment, hardware, such as the NI **214**, manages the address mapping.

The OS **212** performs virtual memory (VM) management through the use of a memory mapping function such as the UNIX OS mmap() function which maps the message buffers **208** in OS memory address space **220** to the application program memory address space **224**. Virtual to physical address translations are therefore handled in the existing OS manner. To enable the OS **212** to perform VM management and address translation, the INCA NI driver **202** must allocate message buffers **208** in the OS memory address space **220** initially, as well as in the application program memory address space **224** to allow the OS **212** to make the required mappings and perform its VM management functions. The INCA NI driver **202** performs these functions as soon as the message arrival signal is received by the INCA NI driver **202**. The address locations of the message buffers **208** containing the network communicated data are therefore mapped to the VM locations in the IPP address space **218**, with only one physical memory location, hence no copying of the network communicated data is required.

Message buffer management and DMA management are performed by the INCA NI driver **202**. The INCA NI driver **202** allocates buffer space when an application **210** calls the INCA NI driver **202** with an INCA open() call which opens the INCA NI driver **202** to initialize the DMA transfer. The INCA NI driver **202** receives the NI message interrupt signal and starts the INCA NI driver **202** which causes message buffer allocation to occur in message buffers **208** and in IPP address space **226**. The INCA NI driver **202** uses the 4 KB memory page size provided by most OS VM systems, and allocates message buffers in 2 KB increments. Each message is aligned to the beginning of the 2 KB buffer with a single message per buffer for

messages smaller than 2 KB. For messages larger than 2 KB, multiple buffers are used beginning with the first and intermediate buffers filled from start to end. The last buffer contains the remnant of the message starting from the 2 KB buffer VM address position. For messages equal to or less than 2 KB, one buffer is allocated and the contents are aligned with the first byte placed at the start of the 2 KB buffer address space.

In order to make the mapping from the OS space to user space easier and in order to avoid implementing more memory management functionality into INCA, the message buffers are "pinned" or assigned to fixed physical memory locations in either the application or OS address space. The application specifies message buffers using offsets in the buffer region, which the INCA NI driver **202** can easily bounds-check and translate. By using fixed physical memory locations, the INCA NI driver **202** will not issue illegal DMA access. Since INCA has complete control over the size, location, and alignment of physical buffers, a variety of buffer management schemes are possible.

All buffers may be part of a system-wide pool, allocated autonomously by each domain (e.g., applications, OS), located in a shared VM region, or they may reside outside of main memory on a NI device. Physical buffers are of a fixed size to simplify and speed allocation. The INCA NI driver memory management is immutable, it allows the transparent use of page remapping, shared virtual memory, and other VM techniques for the cross-domain transfer of network communicated data. Virtual copying with the mmap() function is used to make domain crossings as efficient as possible, by avoiding physical memory bus transfer copying between the OS **212** and application program memory address space **224**.

The third function of the INCA NI driver **202** is message demultiplexing (for receiving) and multiplexing (for transmitting). Not all applications on a machine may be using the INCA software to communicate over the network. There may be a mix of INCA and non INCA communicating applications in which case the INCA NI driver **202** must also route messages to

the non INCA NI driver or the non INCA protocol processing software, or to some other non INCA software. The INCA NI driver **202** maintains a list of INCA application program addresses known as endpoints. Endpoints provide some of the information required to carry out the INCA NI driver component functions.

Referring to **Figure 3**, an overview of the endpoint mechanism is shown. Endpoints **302** bear some resemblance to conventional sockets or ports. A separate endpoint is established and maintained for each application and each network connection for each application. For applications without INCA endpoint addresses, non INCA networking applications, the INCA NI driver passes the message arrival notification to the non INCA NI driver.

Each application that wishes to access the network first requests one or more endpoints **302** through the INCA alternative API "system calls". The INCA NI driver then associates a set of send **304**, receive **306**, and free **308** message queues with each endpoint through the use of two INCA "system calls", inca_create_endpoint() and inca_create_chan(). The application program memory address space **300** contains the network communicated data and the endpoint message queues (endpoint send/receive free queues **304, 306, 308**) which contain descriptors for network messages that are to be sent or that have been received.

In order to send, an application program composes a network message in one or more transmit buffers in its address space and pushes a descriptor onto the send queue **304** using the INCA API "system calls". The descriptor contains pointers to the transmit buffers, their lengths and a destination tag. The INCA NI driver picks up the descriptor, allocates virtual addresses for message buffers in OS memory address space and sets up DMA addresses. The INCA NI driver then transfers the network communicated data directly from the application program memory address space message buffers to the network. If the network is backed up, the INCA NI driver will simply leave the descriptor in the queue and eventually notifies the user application process to slow down or cease transmitting when the queue is near full. The INCA

NI driver provides a mechanism to indicate whether a message in the queue has been injected into the network, typically by setting a flag in the descriptor. This indicates that the associated send buffer can be reused.

When the INCA NI driver receives network communicated data, it examines the message header and matches it with the message tags to determine the correct destination endpoint. The INCA NI driver then pops free buffer descriptors off the appropriate free queue **308**, translates the virtual addresses, transfers the network communicated data into the message buffers in OS memory address space, maps the memory locations to the application program memory address space and transfers a descriptor onto the receive queue **306**. Each endpoint contains all states associated with an application's network "port".

Preparing an endpoint for use requires initializing handler-table entries, setting an endpoint tag, establishing translation table mappings to destination endpoints, and setting the virtual-memory segment base address and length. The user application program uses the API routine calls "ioctl()" and "mmap()" to pass on any required endpoint data and provide the VM address mapping of the OS message buffers to the application program memory address space locations. Once this has been achieved, the user application is prepared to transmit and receive network communicated data directly into application program memory address space. Each endpoint **302** is associated with a buffer area that is pinned to contiguous physical memory and holds all buffers used with that endpoint. Message descriptors contain offsets in the buffer area (instead of full virtual addresses) which are bounds-checked and added to the physical base address of the buffer area by the INCA NI driver. In summary, endpoints and their associated INCA NI driver "system calls" set up an OS-Bypass channel for routing network communicated data address locations to and from memory to the correct applications.

Providing some security is the fourth function performed by the INCA NI driver. To assure that only the correct applications access the message data, application program identifiers

to endpoints and endpoints to message data mappings are maintained. An application can only access message data in the endpoint message queues where the identifiers of endpoint(s) of message queues matches the identifiers of endpoints for that application. Any access to network communicated data must come from the intended recipient application or in the case of transmitting network communicated data, access to the network communicated data must come from the transmitting application.

Once the network communicated data transfer is set up and demultiplexing of messages is complete, the INCA NI driver performs the function of transferring the network communicated data from the OS memory address space to the receiving application program memory address space. This transfer is required since all present NI devices come under the ownership of the computer OS and any network communicated data transferred via a NI device is allocated to the OS virtual or physical memory address space. INCA makes this transfer without requiring any movement or copying of the network communicated data, thereby avoiding costly data copying. The transfer is made via a mapping of the memory addresses of the network communicated data within the OS memory address space to memory (addresses) within the application program memory address space.

For UNIX OS based systems, the UNIX mmap() function is used by the INCA NI driver to perform the transferring of network communicated data to the application address space, mapping the addresses of the network data in the OS address space to the application address space.

The sixth function of the INCA NI driver is to interface to INCA's second component, the IPP loop software. Once network communicated data is available in the computer's memory, the INCA NI driver notifies the IPP software that network communicated data is available for protocol processing. The notification includes passing a number of parameters to provide needed details for the IPP software. The parameters include the addresses of the network communicated

data and the endpoints to determine the recipient application program.

The IPP component of the invention is an extension of Integrated Layer Processing (ILP), performing the functions of communications protocol processing. IPP includes protocols above the transport layer, including presentation layer and application layer protocol processing and places the ILP loop into one integrated execution path with the INCA NI driver and API software. In current systems, communication protocol processing is conducted as a part of and under the control of the OS in OS memory address space. Each protocol is a separate process requiring all the overhead of non integrated, individual processes executed in a serial fashion. Existing and research implementations do not integrate ILP with an NI OS-Bypass message handler and driver, do not integrate protocol processing into a single IPP loop, nor do they execute protocols in user application program memory space. Protocol execution by the existing OSs and under the control of the OS are not used by INCA's IPP component. INCA's IPP performs protocol execution using the INCA software library implementations of the protocols linked to the application in the application program memory address space.

Referring to **Figure 4a**, a depiction of a "C" code example of typical protocol processing code is shown. Before the code can be executed, the network communicated data must be copied to each protocol's memory address space. When the code is compiled to run on a reduced instruction set computer (RISC) CPU, the network message data manipulation steps results in the machine instructions noted in the comments. First, the protocol software process, e.g., the Internet Protocol (IP) software, is initialized and the network communicated data is copied from the OS message buffer memory area to the IP process execution memory area. Each time a word of network communicated data is manipulated, the word is loaded and stored into memory. Upon completion of the first protocol, the second protocol process, e.g., the TCP software, is initialized and the network communicated data is copied to this protocol's execution area in memory. Once again, each time a word of network communicated data is manipulated, the word

is loaded and stored. This process continues until all protocol processing is complete.

Referring to **Figure 4b**, the INCA system with the IPP method is shown, where each word is loaded and stored only once, even though it is manipulated twice. Each protocol's software execution loop is executed in one larger loop, eliminating one load and one store per word of data. This is possible because the data word remains in a register between the two data manipulations. Integrating the protocol processing for-loops results in the elimination of one load and one store per word of network communicated data. The IPP method of performing all protocol processing as one integrated process, also eliminates the copying of all network communicated data between the initialization and completion of each separate communications protocol used (e.g., copying the data to IP protocol address space, then copying the data to UDP or TCP protocol address space, etc.).

In addition, the INCA IPP protocol processing uses an optimized protocol checksum processing routine that calculates checksums on a word (e.g., 4 to 8 bytes depending upon the machine) of network communicated data at a time, rather than the existing method of one byte at a time. INCA's IPP checksum calculation is roughly four times faster than existing checksum calculations. For small message sizes of less than or equal to 200 bytes, which comprise some 90% or more of all network messages, INCA's IPP checksum routine greatly speeds up the processing of small messages since checksum calculation is the majority of calculations required for small messages.

The IPP component divides protocol processing of network messages into three categories: data manipulation - reading and writing application data, header processing - reading, writing headers and manipulating the headers of protocols that come after this protocol, and external behavior - passing messages to adjacent layers, initiating messages such as acknowledgments, invoking non-message operations on other layers such as passing congestion control information, and updating protocol state such as updating the sequence number

associated with a connection to reflect that a message with the previous number has been received.

Referring to **Figure 5**, the INCA IPP component executes the protocols in three stages in a processing loop: an initial stage **502**, a data manipulation stage **504** and a final stage **506**. The initial stages of a series of layers are executed serially, then the integrated data manipulations take place in one shared stage and then the final stages are executed serially. Interoperability with existing protocol combinations such as IP, TCP, UDP and External Data Representation (XDR) combinations requires the IPP software to contain some serial protocol function processing of the network communicated data in order to meet the data processing ordering requirements of these existing protocols. Message processing tasks are executed in the appropriate stages to satisfy the ordering constraints. Header processing is assigned to the initial stage. Data manipulation is assigned to the integrated stage. Header processing for sending and external behavior (e.g., error handling) are assigned to the final stage.

Referring to **Figures 5** and **6**, INCA's IPP method of integrating multiple protocols is shown. The protocols of protocol A **610** and protocol B **620** are combined and INCA's IPP method integrates the combination of protocol A and B **628**. The initial stages **612, 622** are executed serially **502** (as shown in Figure 5), then the integrated data manipulation **504** is executed **614, 624**, and then the final stages **616, 626** are executed serially **506**. Executing the tasks in the appropriate stages ensures that the external constraints protocols impose on each other cannot conflict with their internal constraints.

The ILP software starts up directly after reception of network communicated data into the message buffer. It does the integrated checksumming on the network communicated data in the initial stage **502**, protocol data manipulations and Host/Network byte order conversions in the middle integrated stage **504**, and TCP type flow control and error handling in the final stage **506**. The concept of delayed checksumming has been included in the loop. In the case of IP

fragments, the checksumming is done only after reassembly. Message fragments are transmitted in the reverse order, i.e., the last fragment is transmitted first, to make the time of checksumming less in the case of UDP. Once the data processing is complete, the packets are multiplexed to the corresponding protocol ports set up by the API.

The IPP protocol library software consists of software functions that implement the protocol processing loop and other pieces of protocol control settings such as fragmentation, and in the case of TCP, maintaining the window, setting up time-out and retransmission etc. The TCP library has been implemented with a timer mechanism based on the real-time clock and a Finite State Machine (FSM) implementation.

The INCA IPP component integrates protocol processing into one process which executes in the application program's memory address space. Consequently, the number of copies of the network communicated data to and from memory are greatly reduced, as can be seen by comparing Figures 1 and 2. Thus the speed and efficiency with which data can be accessed and used by an application is increased. These repeated transfers across the CPU/memory data path frequently dominate the time required to process a message, and thus the network communications throughput. The IPP component therefore speeds up network communicated data processing through the elimination of time consuming memory bus transfers of the network communicated data. With the use of the IPP loop's high performance protocol checksum software, protocol processing time of network communicated data is greatly reduced, providing a large part of the performance improvement of the invention.

Interoperability requires the IPP loops to use and execute existing protocols as well as future protocols. The INCA software libraries accommodate the integration of all existing protocols and future protocols into the IPP component of INCA and provide integration with the INCA NI driver component functions.

The seventh function of the INCA NI driver is to interface to INCA's third component,

the API. This interface provides the application with network access for sending data and also sets up application address space parameters for transfer of network data to an application.

The API component of the invention provides the interface between the existing application programs and the new INCA components. The API allows existing applications to call on the INCA software to perform network communicated data handling in the new, high performance manner. The API limits the changes required to existing application programs to minor name changes to their current API and thereby provides interoperability with existing application programs. The INCA API allows the application to: open a network connection by opening the NI device, specify parameters to the INCA NI driver, specify the protocols to use and their options, set the characteristics of the data transfer between the application and the network using IPP and the INCA NI driver, and detect the arrival of messages by polling the receive queue, by blocking until a message arrives, or by receiving an asynchronous notification on message arrival (e.g., a signal from the INCA NI driver). The API also provides low level communication primitives in which network message reception and transmission can be tested and measured.

Referring to **Figure 7**, the INCA API uses alternative "system call" type programming code structures **701** to **712** in place of the current OS system calls such as socket(), connect(), listen() and bin(). The alternative calls are used to bypass the current OS system calls. In an alternative embodiment, the operating system can include the alternative system calls. The new system calls initiate the INCA IPP and INCA NI driver software library programs to provide the necessary API functionality. The API set of system calls **701** to **712** simplifies the application programming required to use the invention, the INCA software library, to renaming the existing calls by placing "inca_" in front of the existing calls. As depicted in **Figure 7**, the API provides the basic commands like "open()", "close()", "read()", "write()", etc., similar to existing system networking APIs.

The "open()" call **701**, **702** and **709** will perform the following for the user:

1. Open the device for operation;
2. Create the OS Bypass structure and set up a DMA channel for user to network transfer;
3. Map the communication segment from the driver buffer to the user buffer;
4. Open an unique channel for communication between two communicating entities;
5. Fill up this buffer (incabuffer), which will be used in calls to read, write, etc.

The "close()" call **703**, **704** and **710** will perform the following for the user:

1. Free storage allocated for the buffers;
2. Destroy the communication channel;
3. Unmap the communication segment;
4. Remove the DMA mapping;
5. Close the device used by that particular application.

The "read()" call **705** and **706**, with a pointer to "incabuffer" as the parameter, will perform the following for the user:

1. Receive any pending packet from the INCA device, which has been transferred via DMA;
2. Pass the received packet (if not fragmented) through the IPP loop; or if fragmented, pass the received packet through a special IPP loop (inlined) which will do IPP, but will keep track of the fragments and pass it on only when the packet is complete and reassembled;
3. Close the read call (the packet is ready for delivery to the application).

The "write()" call **707** and **708**, with a pointer to "incabuffer" as the parameter, will perform the following for the user:

1. Create the header for IP/UDP/TCP;

2. Perform IPP processing on the packet;

3. Fragment the packet if it is too large for UDP or IP by passing the received packet through a special IPP loop (inlined), which will keep track of the fragments and pass it on only when the packet fragmentation is complete;

4. Pass on the IP packets for transmission onto the NI.

Parameters passed by the application to the IPP and NI driver components of INCA inside the () of the calls include application message identifiers and endpoints. This parameter allows the IPP and INCA NI driver components to multiplex and demultiplex messages to the intended applications or network address. A more enhanced API could include calls or parameters within calls to set up a connection using TCP and also functions that help in the implementation of client/server applications like "listen" etc.

Although INCA's API can be located anywhere between the networking application program and any protocol of the protocol stack, the API is typically located between the application and the INCA IPP component. In current OSs, the API typically sits between the session layer (e.g., socket system calls) and the application. Ideally, the API sits between the application and all other communication protocols and functions. In current systems and application programs, many times the application also contains the application layer protocol (i.e., Hypertext Transport Protocol - HTTP), the presentation layer protocol functions (i.e., XDR like data manipulation functions), and only the socket or streams system call is the API. This is not necessarily ideal from a user perspective. By integrating presentation and application protocol functions into the application, any change in these functions necessitates an application program "upgrade" at additional procurement, installation time and maintenance cost. INCA can incorporate all the application, presentation and session (replacement for socket and streams calls) functions into the IPP loop. In the future, this can even be accomplished dynamically, at run time, through application selection of the protocol stack configuration.

The API provides the application the link to utilize the INCA high performance network communication subsystem as opposed to using the existing system API and existing network communicated data processing subsystem. The existing system API could also be used to interface to the INCA IPP and INCA NI driver components if the existing system API is modified to interface to the INCA IPP protocol processing and INCA NI driver network interface and instead of the existing system protocol processing and network interface software.

The final function is relinquishing control of the NI device. The INCA NI driver uses an alternative "system call", inca_closedev(), in place of the current OS system call to close the NI device and to relinquish control of the NI device. When the NI device has no more network communicated data to be transferred, the INCA NI driver relinquishes control of the NI device to the computer's OS so that other software can use the NI device. Hardware or software interrupts are typically used to signal that the NI device has no more network communicated data to transfer. Upon detection of the no more network communicated data to transfer signal, the INCA NI driver sets the end memory address of the network communicated data buffers. For the mapping of the network communicated data into the application address space, the INCA NI driver performs any required message buffer alignment and passes the network communicated data address range to the IPP software. The NI device is set to a known state and the OS is notified that the device is available to be scheduled for use by other software processes.

To illustrate the workings and ease of use of the invention, the following description is provided. The INCA software library is loaded unto the computer's hard disk. If the machine's NI device drivers are implemented as loadable modules, no NI device driver modifications are necessary. If the INCA NI driver is integrated into the OS without being a separate module, the INCA NI driver software is integrated into the OS through a recompilation of the OS. This does not alter the operation of existing programs or the OS, but only adds the INCA NI interface. For those networking applications that will use the INCA software library for network message

handling, the API system calls are changed to the INCA API system calls. This procedure can be accomplished via a number of approaches. Once accomplished, all is complete and system operation can resume. These two steps, INCA NI driver insertion and renaming the API calls, provide a system by system approach to using the INCA networking software. System vendors or system administrators are the most likely candidates to use this method. An alternative approach to the above steps is to integrate the entire library into the applications desiring to perform networking with the INCA software. This provides an application by application approach to using INCA. Application program vendors or individual users are the most likely candidates to use this method. Either way, the entire procedure can be accomplished in minutes to hours depending upon the implementor's familiarity with the applications, OS and NI device drivers. For existing applications that do not have their system calls modified, INCA allows the traditional network system interfaces (e.g., sockets) with the applications. Referring to **Figure 11**, the order of events for receiving network communicated data over the network is shown. Once system operation begins and network messages are received, the order of events for receiving data over the network are as follows: the NI device driver receives a message arrival notification from the NI hardware **1100**, typically via a hardware interrupt. The message arrival notification signal is received by the OS and initiates the opening of the INCA enhanced NI driver - the INCA NI driver **1102**. The INCA NI driver determines if the network message is for an application that can use the INCA software library to receive messages **1104**. If the application is not "INCA aware," control is handed over to the OS for further handling **1108**. If the application can use INCA to communicate, the INCA NI driver takes control of the NI device **1106** (e.g., Asynchronous Transfer Mode - ATM network card) and sets up the registers, firmware, etc., of the device to transfer the network communicated data from the NI device to internal computer memory **1110**. The INCA NI driver uses an alternative "system call" type programming code structure, inca_opendev(), in place of the current OS system calls to take over

the device and set up the data transfer from the NI device to computer memory. The INCA driver then uses the endpoint identifiers to demultiplex incoming messages to the recipient application program **1112**. The network message buffers in OS address space are mapped to the recipient application's address space **1114**. The INCA IPP software is configured and started for protocol processing **1116**. The IPP software performs protocol processing to extract the data from the network message(s) **1118**. Once the first IPP loop is completed, the application is notified via the INCA API calls that data is ready for consumption **1120**. The application then processes the data **1122**. If there are more messages to process, the IPP loop continues processing and the application continues consuming the data **1124**. When all messages have been received **1126**, the NI driver closes the NI device and relinquishes control of the device to the OS **1128**.

For transmission of data, the entire process occurs in reverse order and the application uses the API calls to communicate with the IPP software to determine which protocols and protocol options to use, sets up an endpoint by opening the INCA NI driver with the open() API call, establishes an endpoint, sets the endpoint and driver DMA characteristics with INCA API system calls such as ioctl(), and upon transmission completion, uses close() to close the INCA NI driver. The IPP component executes the selected protocols and places the resulting network communicated data into the send queue message buffers. The INCA NI driver ceases control of the NI device and DMA resources with its "system calls" to the OS, maps the send queue in application address space to the OS message buffers in OS address space using the function mmap(), sets up and controls the DMA transfer from the OS message buffers to the NI device, and upon completion, relinquishes control of the NI device and DMA resources.

The API calls are the method of communication between the three INCA components and the existing application programs, OS and computer resources such as the NI and DMA devices.

RESULTS

Tests were conducted on commercially available systems, configured with commercial-off-the shelf (COTS) software, NIs and a FastEthernet network. The INCA testbed consisted of two machines connected via a 100 Mbps FastEthernet. INCA allows applications to process data at rates greater than 10 Mbps, thereby a normal 10 Mbps Ethernet would have caused the network to limit INCA performance. A SUN Microsystems UltraSPARC1 WS with a 143 MHz UltraSPARC1 CPU, 64 MB of RAM, running Solaris 2.5.1 (also known as SUN OS 5.5.1), with a SUN SBus FastEthernet Adapter 2.0 NI was connected via a private (no other machines on the network) 100 Mbps FastEthernet to a Gateway 2000 PC with a 167 MHz Pentium Pro CPU, 32 MB of RAM, running the Linux 2.0 OS with a 3Com FastEtherlink (Parallel Tasking PCI 10/100 Base-T) FastEthernet NI. Messages of varying lengths from 10 bytes to the maximum allowable UDP size of 65K bytes were sent back and forth across the network between the machines using an Internet World Wide Web (WWW) browser as the application program on both machines. This architecture uses the actual application programs, machines, OSs, NIs, message types and networks found in many computing environments. The results should therefore have wide applicability.

SUN UltraSPARC1 Workstation with and without INCA

Referring to **Figure 8**, the graph illustrates the fact that on a high performance WS class computer, INCA outperforms the current system at application program network message throughput by 260% to 760% depending upon the message size. Since 99% of TCP and 89% of UDP messages are below 200 bytes in size, the region of particular interest is between 20 and 200 byte size messages.

Gateway 2000 Pentium Pro PC with and without INCA

Referring to **Figures 9** and **10**, the graphs illustrate that on a PC class computer, INCA outperforms the current system at application program network message throughput by 260% to 590%. **Figure 9** shows INCA's 260% to 275% performance improvement for message sizes

of 10 to 200 bytes. **Figure 10** shows that as message sizes get larger and larger, up to the existing protocol limit of 65K bytes, INCA's performance improvement becomes larger and larger reaching the maximum of 590% at a message size of 65K bytes.

Although the method of the present invention has been described in detail for purpose of illustration, it is understood that such detail is solely for that purpose, and variations can be made therein by those skilled in the art without departing from the spirit and scope of the invention. The method of the present invention is defined by the following claims:

We claim:

1. A method for improving the internal computer throughput rate of network communicated data comprising transferring network communicated data from a network interface device to an application address space with only one physical copying of the data.

2. The method for improving the internal computer throughput rate of network communicated data of claim 1, wherein the copying of the data occurs in response to a call by an application program interface which bypasses the operating system calls.

3. The method for improving the internal computer throughput rate of network communicated data of claim 2, where the call functions of the application program interface are integrated into the existing operating system.

4. The method for improving the internal computer throughput rate of network communicated data of claims 1 or 2, further comprising the transfer of network communicated data from the network interface device directly to the application address space.

5. The method for improving the internal computer throughput rate of network communicated data of claim 4, further comprising the transfer of network communicated data from the network interface device to application address space through an address mapping of the network communicated data between the operating system address space and the application address space.

6. The method for improving the internal computer throughput rate of network communicated data of claim 5, wherein the transfer of network message data from the network interface device to the application address space is controlled by the operating system.

7. The method for improving the internal computer throughput rate of network communicated data of claim 5, wherein the transfer of network message data from the network interface device to the application address space is controlled by the application program.

8. The method for improving the internal computer throughput rate of network communicated

data of claim 5, wherein the transfer of network message data from the network interface device to the application address space is controlled by a hardware component.

9. The method for improving the internal computer throughput rate of network communicated data of claim 5, wherein the transfer of network message data from the network interface device to the application address space is controlled by the network interface device.

10. The method for improving the internal computer throughput rate of network communicated data of claim 5, wherein the transfer of network message data from the network interface device to the application address space is a direct memory access transfer.

11. The method for improving the internal computer throughput rate of network communicated data of claim 10, further comprising reinitializing a direct memory access if an error occurs.

12. The method for improving the internal computer throughput rate of network communicated data of claim 10, further comprising repeating a direct memory access transfer if an error occurs.

13. The method for improving the internal computer throughput rate of network communicated data of claims 5, wherein the transfer of the network communicated data is a programmed input/output transfer.

14. The method for improving the internal computer throughput rate of network communicated data of claim 5, wherein the operating system of the computer manages the address mapping between the virtual memory addresses and physical memory addresses of the network communicated data in the operating system and application memory address spaces.

15. The method for improving the internal computer throughput rate of network communicated data of claim 14, further comprising the network interface driver translating the address mapping between the virtual memory addresses and physical memory addresses of the network communicated data in the operating system and application memory address spaces.

16. The method for improving the internal computer throughput rate of network communicated data of claim 14, further comprising the network interface driver demultiplexing network

messages and routing the network messages to the proper application.

17. The method for improving the internal computer throughput rate of network communicated data of claim 16, further comprising the network interface driver examining the header of the message to determine the correct destination point of the message.

18. The method for improving the internal computer throughput rate of network communicated data of claim 16, further comprising the network interface driver maintaining a list of the application endpoints.

19. The method for improving the internal computer throughput rate of network communicated data of claim 14, further comprising the network interface driver providing security by permitting only an intended recipient of the network communicated data to access the network communicated data.

20. The method for improving the internal computer throughput rate of network communicated data of claim 14, further comprising the network interface driver notifying and providing parameters to an integrated protocol processing loop to allow an integrated protocol processing loop to perform protocol processing on the network communicated data.

21. The method for improving the internal computer throughput rate of network communicated data of claim 20, wherein the network interface driver sets end memory addresses of the message buffers, aligns the message buffers and passes the range of the message buffers to the integrated protocol processing loop.

22. A method for improving the internal computer throughput rate of network communicated data comprising executing communication protocols in an integrated protocol processing loop.

23. The method for improving the internal computer throughput rate of network communicated data of claim 22, further comprising linking the proper protocols to an application in the application program memory address space.

24. The method for improving the internal computer throughput rate of network communicated

data of claim 22, further comprising the integrated protocol processing loop containing iterations of serial and integrated data manipulations.

25. The method for improving the internal computer throughput rate of network communicated data of claim 22, wherein header processing is performed during serial data manipulation, data manipulation is performed during integrated data manipulation and header and external behavior is performed during serial data manipulation.

26. A method for improving the internal computer throughput rate of network communicated data comprising calculating communication protocol checksums one computer word at a time within an integrated protocol processing loop.

27. The method for improving the internal computer throughput rate of network communicated data of claim 26, wherein the size of a computer word is 32 bits.

28. The method for improving the internal computer throughput rate of network communicated data of claim 26, wherein the size of a computer word is 64 bits.

29. A method for improving the internal computer throughput rate of network communicated data comprising:

transferring network communicated data from a network interface device to an application address space with only one physical copying of the data;

executing communication protocols in an integrated protocol processing loop;

calculating communication protocol checksums one computer word size of data at a time within the integrated protocol processing loop; and

address mapping of the data occurs in response to call functions, where the operating system's calls are bypassed.

30. The method for improving the internal computer throughput rate of network communicated data of claim 29, wherein the call functions are call functions of an application program interface.

31. The method for improving the internal computer throughput rate of network communicated

data of claim 29, wherein the call functions are call functions of an application program.

32. The method for improving the internal computer throughput rate of network communicated data of claim 29, wherein the call functions are call functions of a network interface device.

33. The method for improving the internal computer throughput rate of network communicated data of claim 29, wherein the call functions are call functions of a network interface driver.

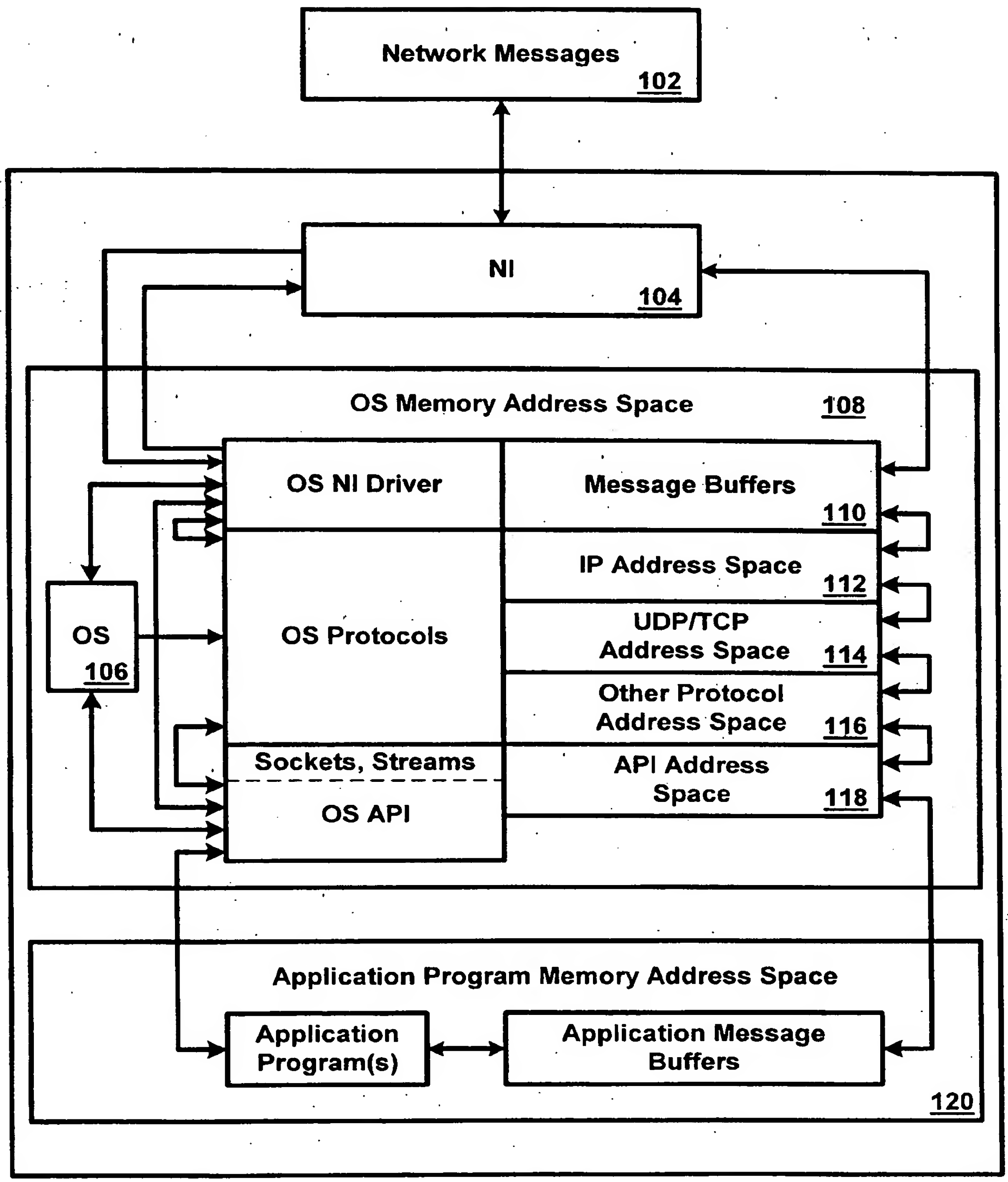

Network Messages
102
NI
104
OS Memory Address Space
108
OS NI Driver
Message Buffers
110
IP Address Space
112
OS
106
OS Protocols
UDP/TCP Address Space
114
Other Protocol Address Space
116
Sockets, Streams
OS API
API Address Space
118
Application Program Memory Address Space
Application Program(s)
Application Message Buffers
120

FIGURE 1

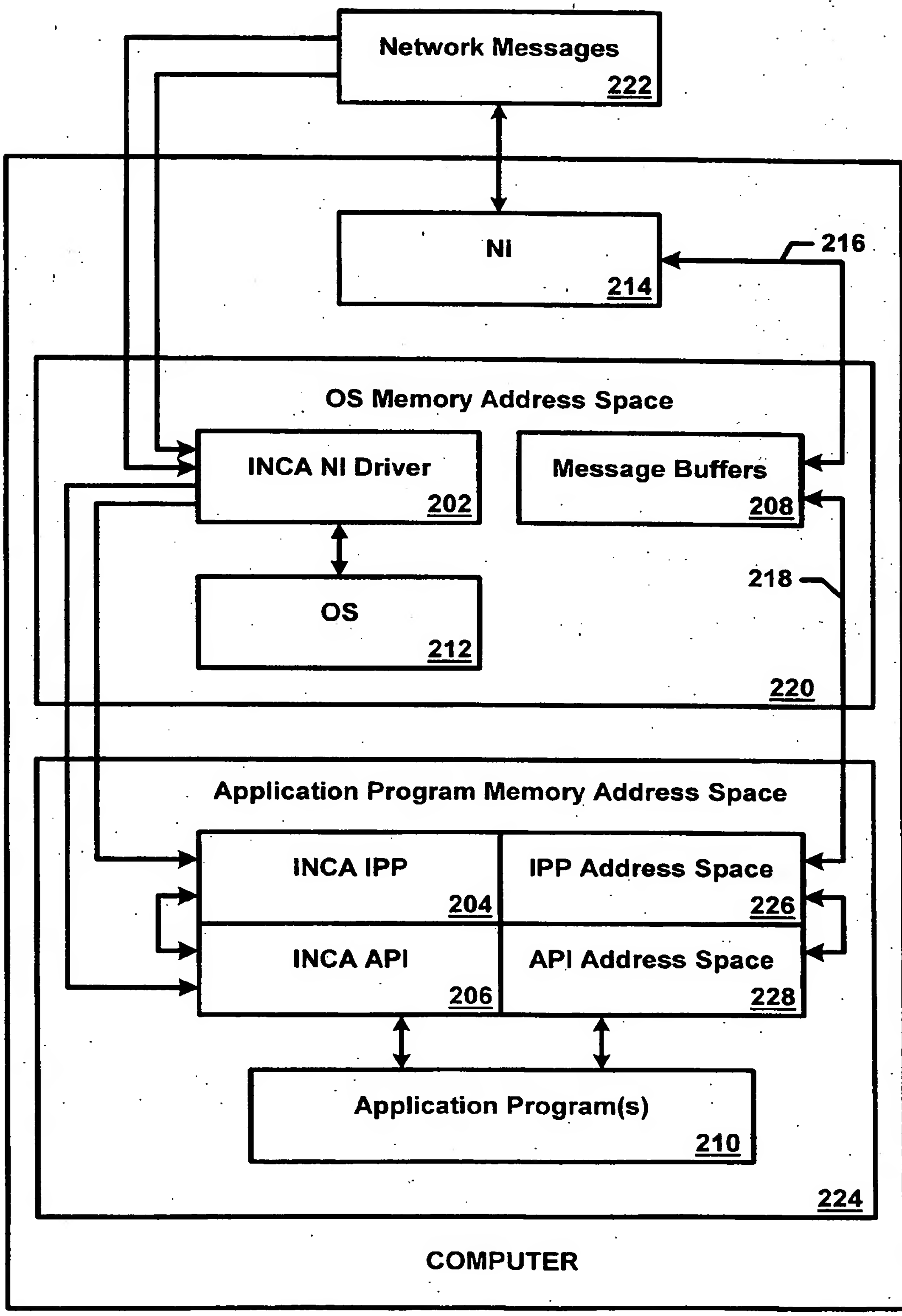

Network Messages
222
NI
214
216
OS Memory Address Space
INCA NI Driver
202
Message Buffers
208
OS
212
218
220
Application Program Memory Address Space
INCA IPP
204
IPP Address Space
226
INCA API
206
API Address Space
228
Application Program(s)
210
224
COMPUTER

FIGURE 2

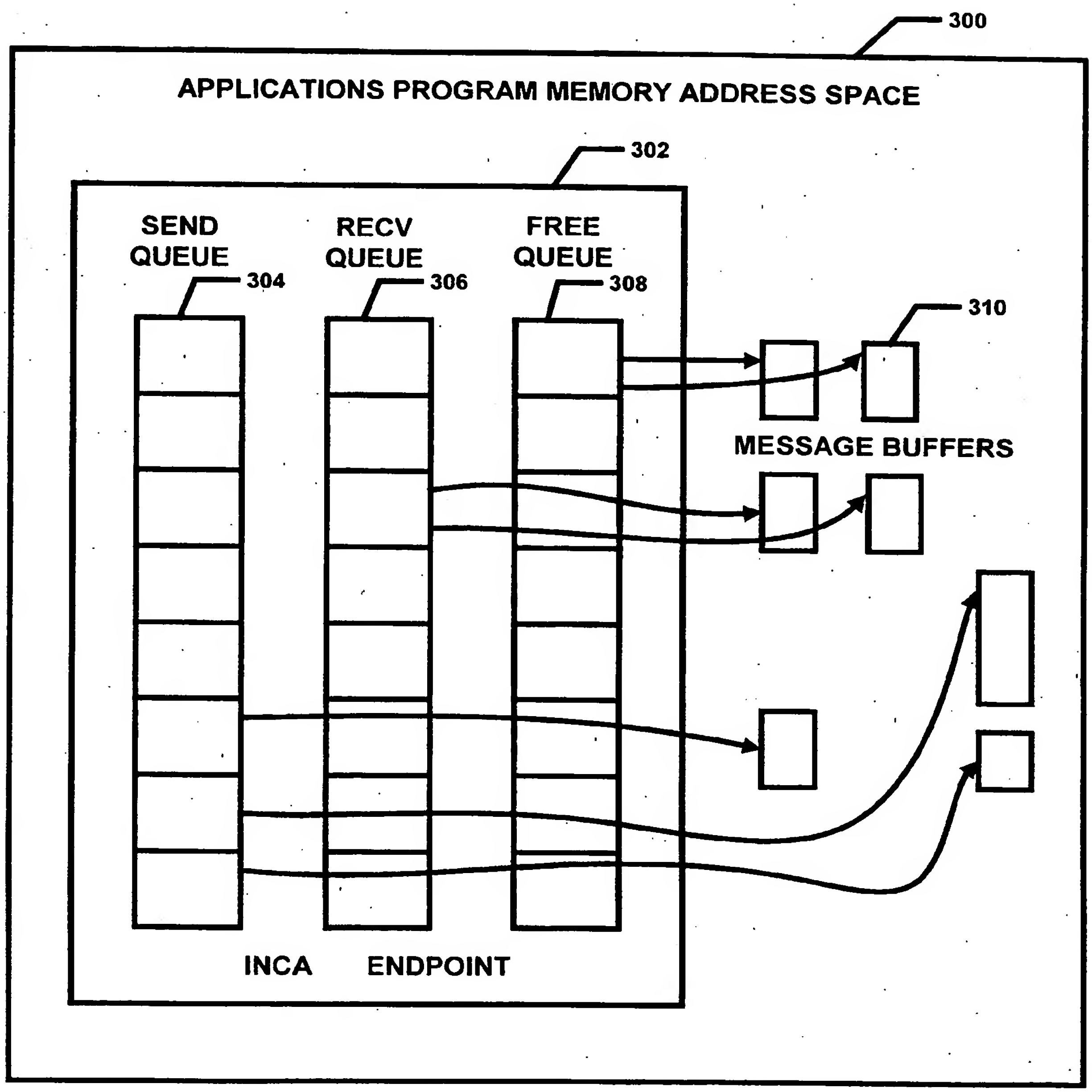

300
APPLICATIONS PROGRAM MEMORY ADDRESS SPACE
302
SEND QUEUE
304
RECV QUEUE
306
FREE QUEUE
308
310
MESSAGE BUFFERS
INCA ENDPOINT

FIGURE 3

```
for( i = 0; i < 1000; i++ )
     msgData[i]++                  /* LOAD, ADD, STORE */
for( i = 0; i < 1000; i++ )
     msgData[i] = ~ msgData[i];    /* LOAD, COMPLEMENT, STORE */
```

FIGURE 4a

```
for( i =0; i < 1000; i++){
     temp = msgData[i];            /* LOAD */
     temp++;                       /* ADD */
     temp = ~ temp;                /* COMPLEMENT */
     msgData[i] = temp;            /* STORE */
          }
```

FIGURE 4b

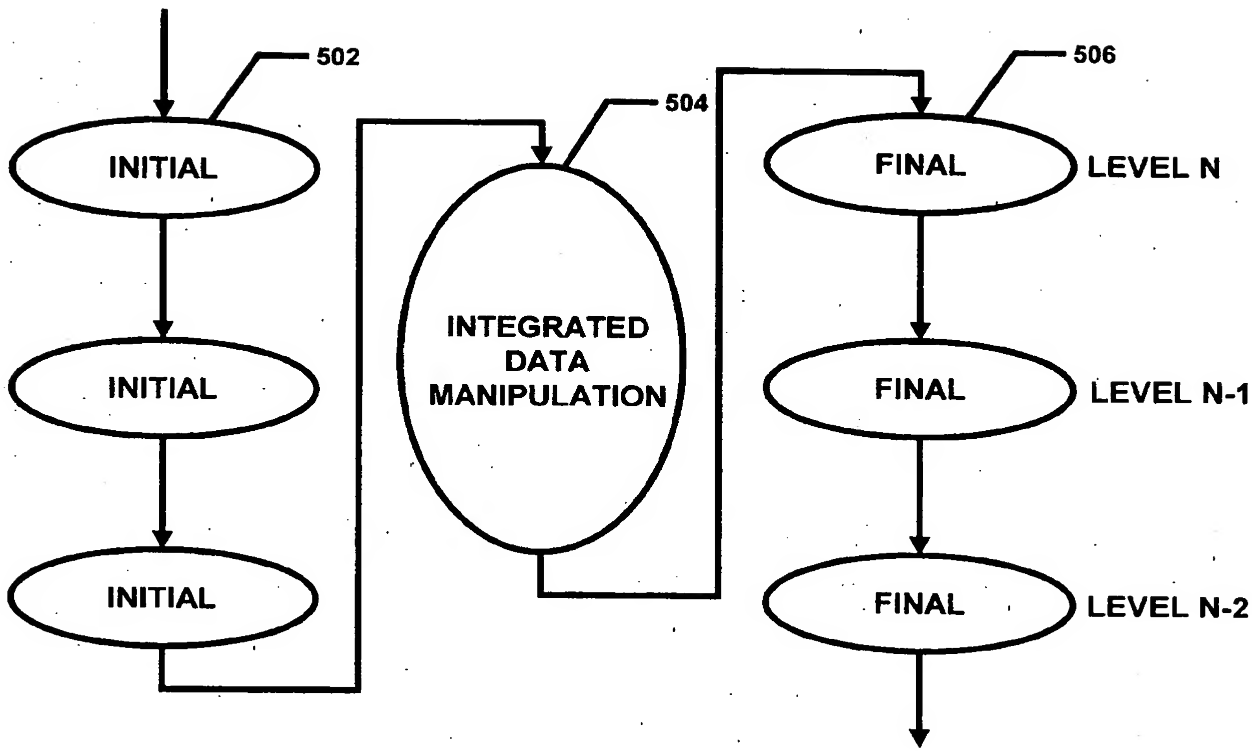
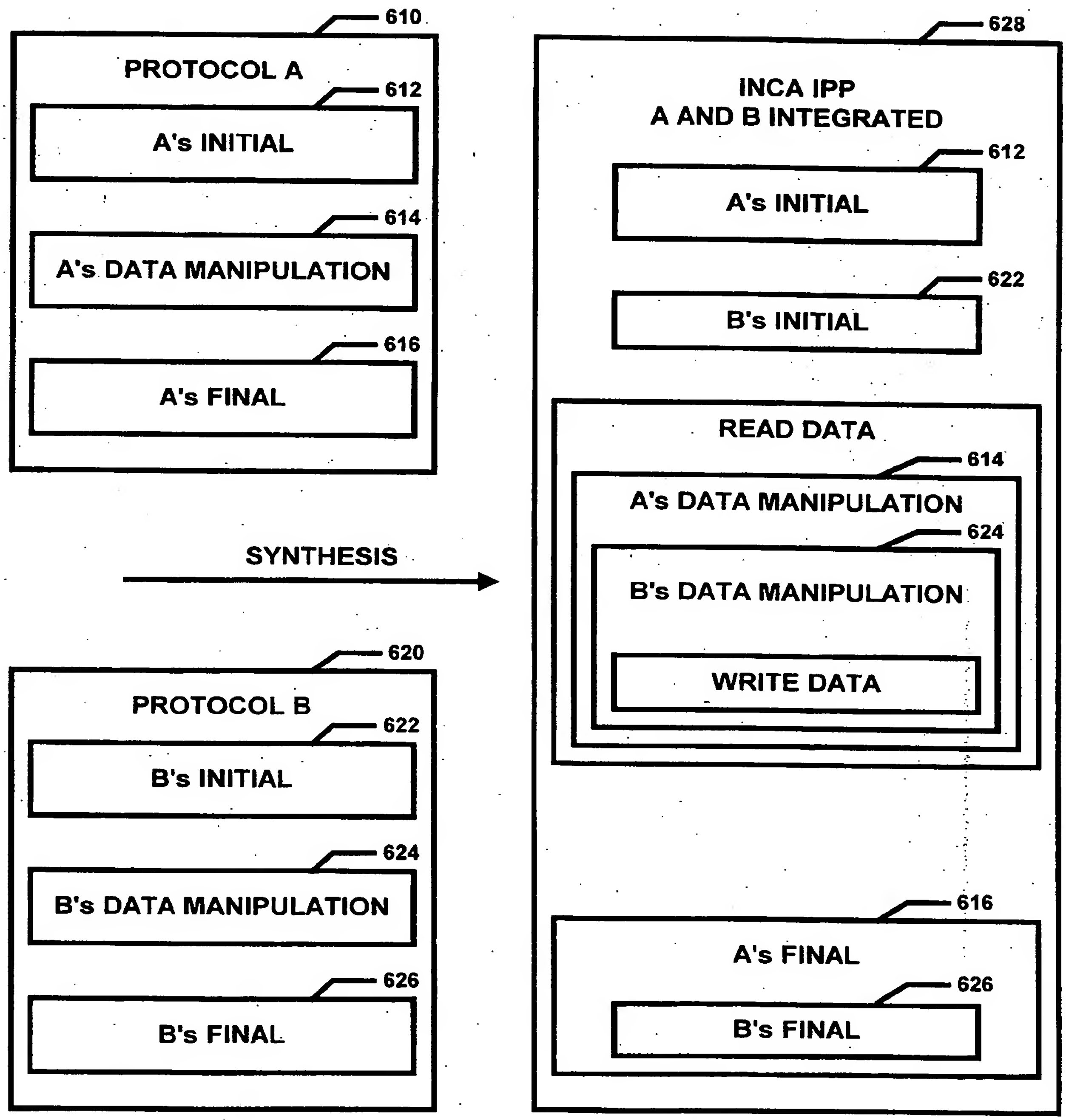

502
INITIAL
INITIAL
INITIAL
504
INTEGRATED DATA MANIPULATION
506
FINAL
LEVEL N
FINAL
LEVEL N-1
FINAL
LEVEL N-2

FIGURE 5



610
PROTOCOL A
612
A's INITIAL
614
A's DATA MANIPULATION
616
A's FINAL
SYNTHESIS
620
PROTOCOL B
622
B's INITIAL
624
B's DATA MANIPULATION
626
B's FINAL
628
INCA IPP
A AND B INTEGRATED
612
A's INITIAL
622
B's INITIAL
READ DATA
614
A's DATA MANIPULATION
624
B's DATA MANIPULATION
WRITE DATA
616
A's FINAL
626
B's FINAL

FIGURE 6

701. inca_open() - open the network device for transmission and set up an OS-Bypass channel for transmission;
702. int inca_open("device name", connector *myconnection) - myconnection: a pointer to a structure connector which is the data structure the API uses for further processing on the opened device; return value: 0 on success, any other number means an error;
703. inca_close() - close the device and destroy the OS-Bypass channel;
704. int inca_close("device-name", connector *myconnection) - return value: 0 on success, any other number means an error;
705. inca_read() - read data from the network through the OS-Bypass channel created; the read performs all the protocol functions and a pointer to the data buffer is returned;
706. int inca_read(connector *myconnection, int length) - connector *myconnection: a pointer to the connector buffer returned by inca_open; myconnection->inca_recv: points to the data buffer received; int length: number of bytes to be read; return value: returns the number of bytes read;
707. inca_write() - write data into the network;
708. int inca_write(connector *myconnection, char *buf, int len) - connector *myconnection: a pointer to the connector buffer returned by inca_open; char *buf: a pointer to the data buffer to be transmitted; int length: the number of bytes to be read.

Used mainly by the NI driver but can be used by the application:

709. inca-opendev() - open the INCA device for use;
710. inca_closedev() - close the device for use;
711. inca_create_endpoint() - create an OS-Bypass endpoint;
712. inca_create _chan() - create a channel for secured multiplexing on the device.

FIGURE 7

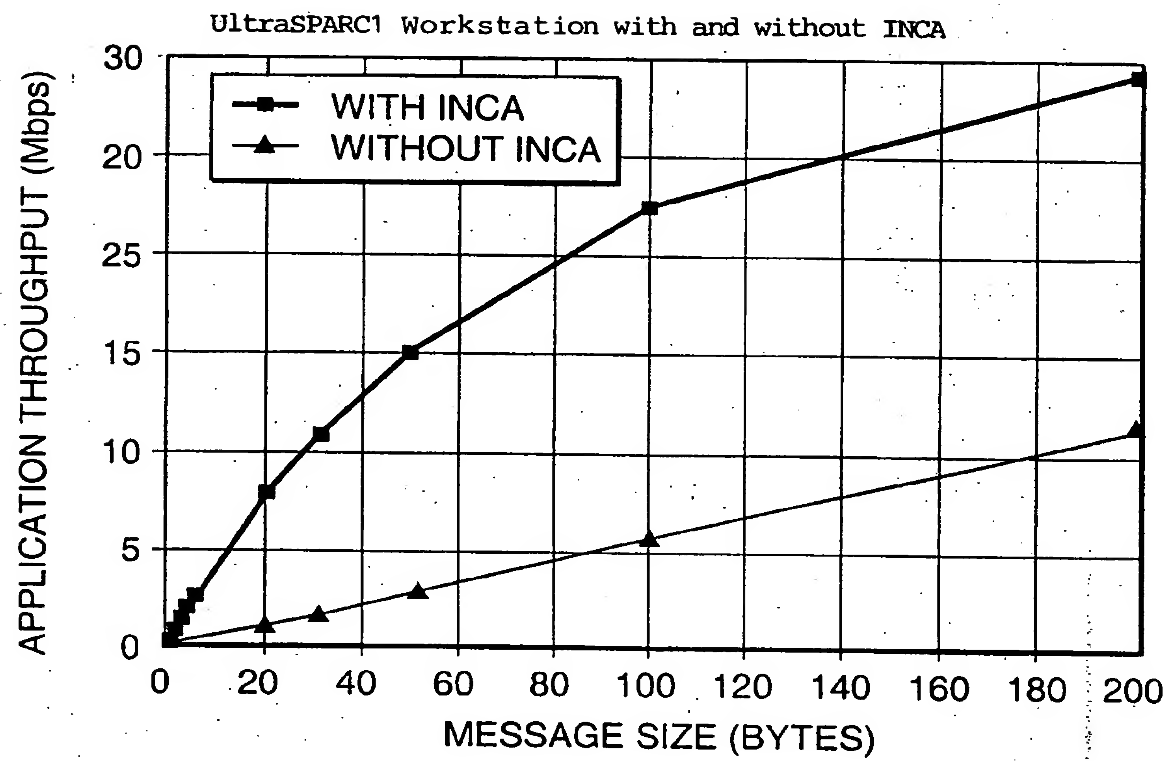

UltraSPARC1 Workstation with and without INCA
WITH INCA
WITHOUT INCA
APPLICATION THROUGHPUT (Mbps)
30
20
25
15
10
5
0
0
20
40
60
80
100
120
140
160
180
200
MESSAGE SIZE (BYTES)

FIG. 8

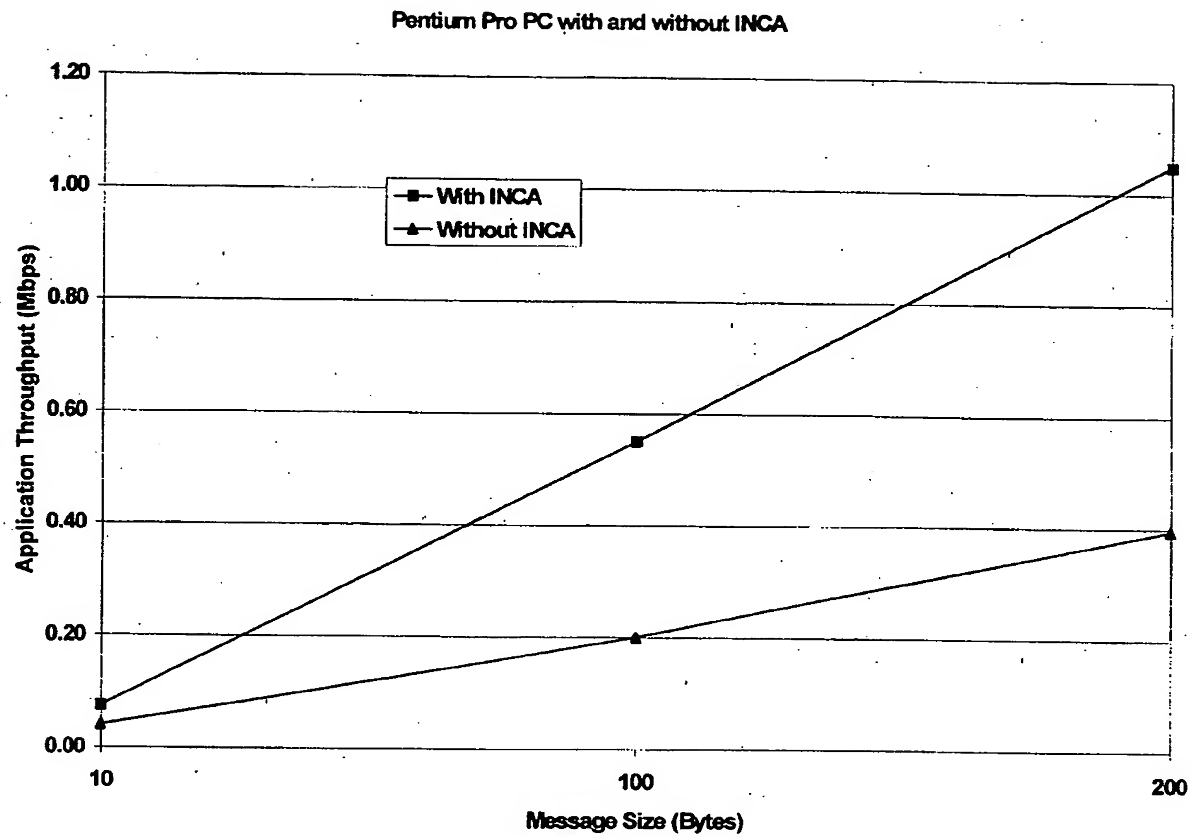

Pentium Pro PC with and without INCA
With INCA
Without INCA
Application Throughput (Mbps)
1.20
1.00
0.80
0.60
0.40
0.20
0.00
10
100
200
Message Size (Bytes)

FIGURE 9

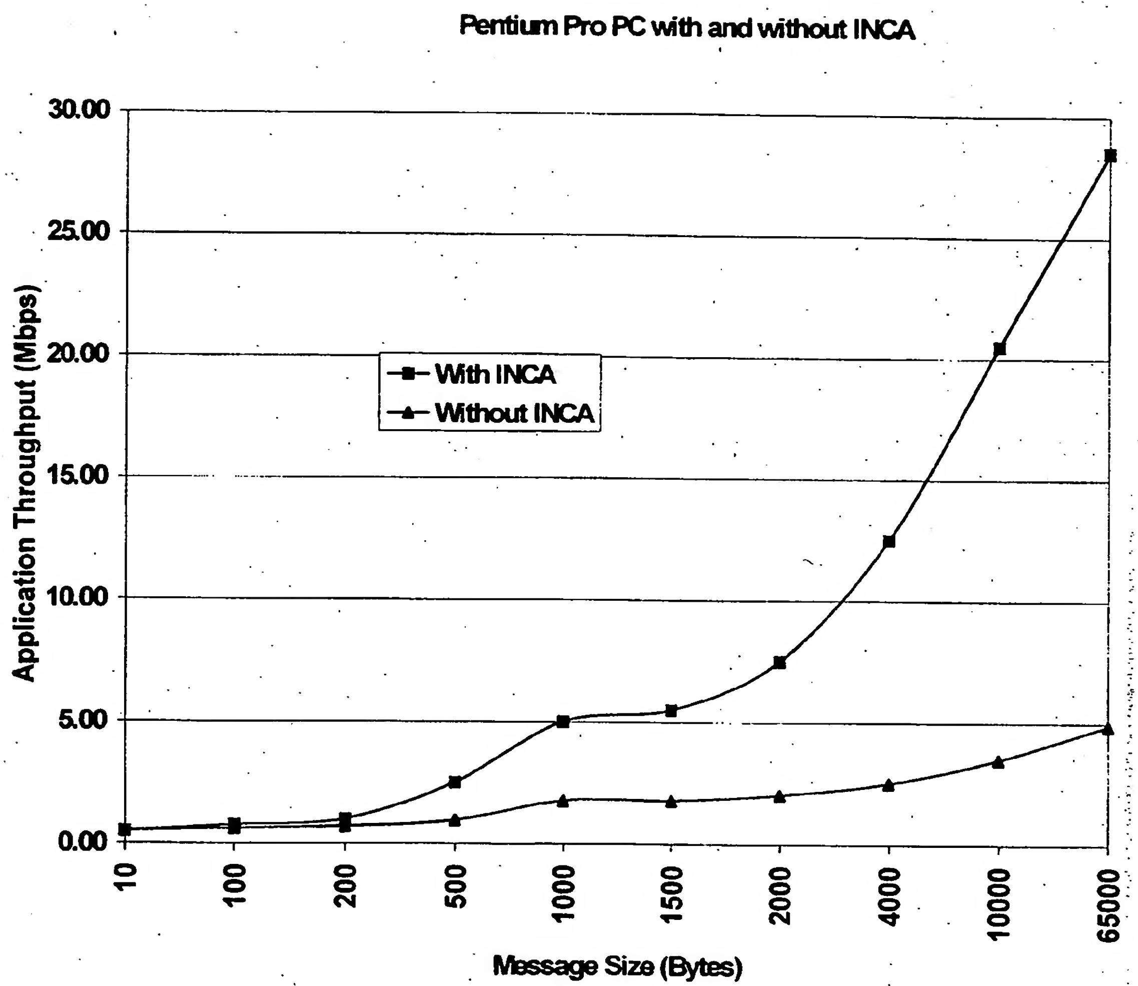
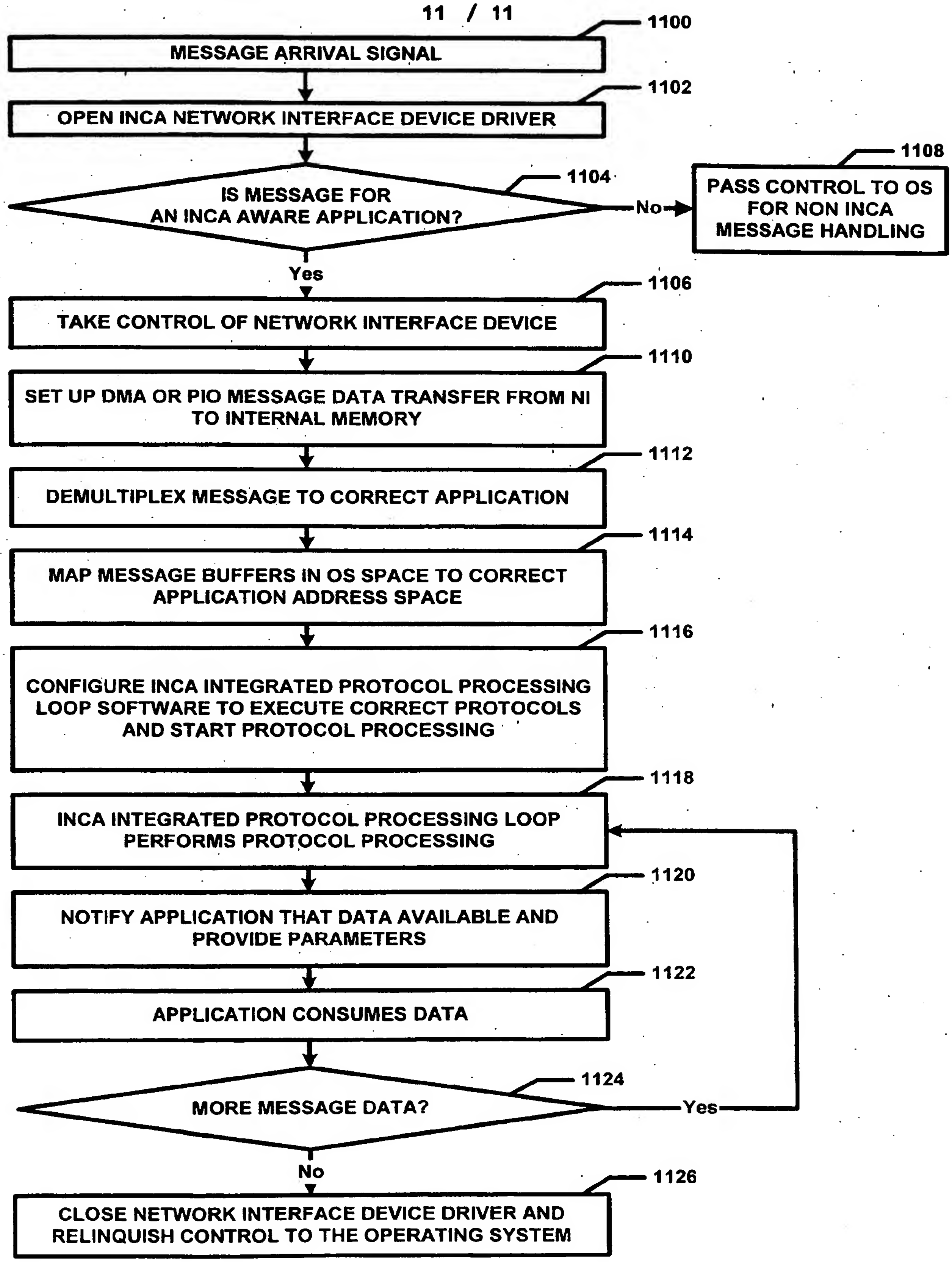

Pentium Pro PC with and without INCA
Application Throughput (Mbps)
30.00
25.00
20.00
15.00
10.00
5.00
0.00
With INCA
Without INCA
10
100
200
500
1000
1500
2000
4000
10000
65000
Message Size (Bytes)

FIGURE 10



1100
MESSAGE ARRIVAL SIGNAL
1102
OPEN INCA NETWORK INTERFACE DEVICE DRIVER
1104
IS MESSAGE FOR AN INCA AWARE APPLICATION?
No
1108
PASS CONTROL TO OS FOR NON INCA MESSAGE HANDLING
Yes
1106
TAKE CONTROL OF NETWORK INTERFACE DEVICE
1110
SET UP DMA OR PIO MESSAGE DATA TRANSFER FROM NI TO INTERNAL MEMORY
1112
DEMULTIPLEX MESSAGE TO CORRECT APPLICATION
1114
MAP MESSAGE BUFFERS IN OS SPACE TO CORRECT APPLICATION ADDRESS SPACE
1116
CONFIGURE INCA INTEGRATED PROTOCOL PROCESSING LOOP SOFTWARE TO EXECUTE CORRECT PROTOCOLS AND START PROTOCOL PROCESSING
1118
INCA INTEGRATED PROTOCOL PROCESSING LOOP PERFORMS PROTOCOL PROCESSING
1120
NOTIFY APPLICATION THAT DATA AVAILABLE AND PROVIDE PARAMETERS
1122
APPLICATION CONSUMES DATA
1124
MORE MESSAGE DATA?
Yes
No
1126
CLOSE NETWORK INTERFACE DEVICE DRIVER AND RELINQUISH CONTROL TO THE OPERATING SYSTEM

FIGURE 11

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/46, H04L 29/06 | A3

(11) International Publication Number: WO 99/26377

(43) International Publication Date: 27 May 1999 (27.05.99)

(21) International Application Number: PCT/US98/24395

(22) International Filing Date: 16 November 1998 (16.11.98)

(30) Priority Data:
08/972,157 17 November 1997 (17.11.97) US

(71) Applicant: MCMZ TECHNOLOGY INNOVATIONS LLC [US/US]; 11029 Grassy Knoll Terrace, Germantown, MD 20876 (US).

(72) Inventor: SCHUG, Klaus, H.; 3360 Gunnison Drive, Fort Collins, CO 80526-2790 (US).

(74) Agents: ROBERTS, Jon, L. et al.; Roberts & Brownell, LLC, Suite 212, 8381 Old Courthouse Road, Vienna, VA 22182 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

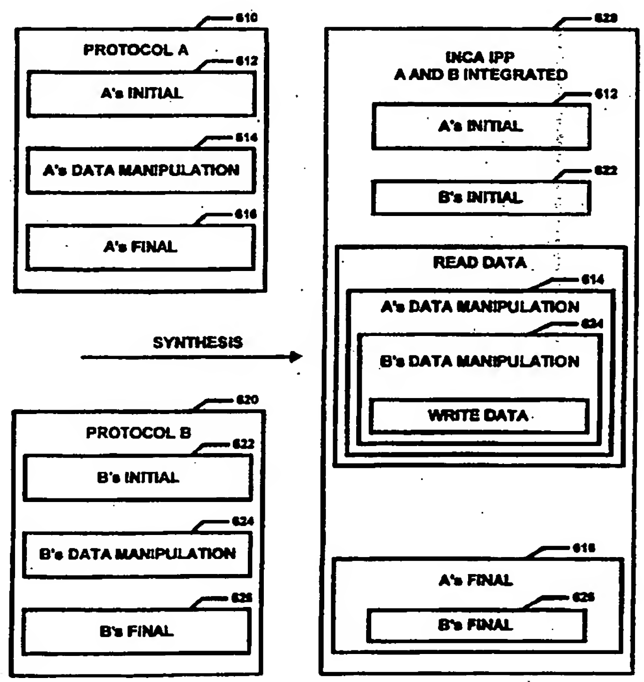
Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(88) Date of publication of the international search report: 16 September 1999 (16.09.99)

(54) Title: A HIGH PERFORMANCE INTEROPERABLE NETWORK COMMUNICATIONS ARCHITECTURE (INCA)

(57) Abstract

An interoperable, software only network communications architecture (INCA) is presented that improves the internal throughput of network communicated data of workstation and PC class computers at the user level, application program level, by 260 % to 760 %. The architecture is unique because it is interoperable with all existing programs, computers and networks requiring minimal effort to set up and use. INCA operates by mapping network data between the application and operating address space without copying the data, integrating all protocol execution into a single processing loop (628) in the application address space, performing protocol checksumming on a machine word size of data within the protocol execution loop, and providing an application program interface very similar to existing application program interfaces. The network interface driver functions are altered to set up network data transfers to and from the application address space without copying of the data to the OS address space, while buffer management, application to message multiplexing/demultiplexing and security functions are also being performed by the modified network interface driver software. Protocols (610, 620) are executed in the application address space in a single integrated protocol processing loop (628) that interfaces directly to the INCA NI driver on one end and to the application on the other end in order to minimize the amount of times that network communicated data must travel across the internal memory bus. A familiar looking application program interface is provided that differs only slightly from existing application program interfaces which allows existing applications to use the new software with a minimum of effort and cost.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 98/24395

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/46 H04L29/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	DRUSCHEL P: "OPERATING SYSTEM SUPPORT FOR HIGH-SPEED COMMUNICATION" COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY, vol. 39, no. 9, September 1996 (1996-09), pages 41-51, XP000642200	1,4-10, 14-17,19
Y	page 42, left-hand column, line 1 - line 5 page 46, right-hand column, line 12 - line 14 page 47, right-hand column, line 11 - line 21 page 47, right-hand column, line 54 - page 48, left-hand column, line 9 page 48, left-hand column, line 52 - line 59 page 48, right-hand column, line 13 - -/--	2,3,13, 18,20,21

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
7 July 1999	02. AUG. 1999
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Masche, C

2

INTERNATIONAL SEARCH REPORT

Int. ...ional Application No
PCT/US 98/24395

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
	page 49, left-hand column, line 22 page 49, left-hand column, line 59 - right-hand column, line 26 figure 3 ---	
Y	EP 0 707 266 A (SUN MICROSYSTEMS INC) 17 April 1996 (1996-04-17)	2,3
A		1,4-7,9, 10,14
	abstract column 2, line 19 - line 20 page 3, line 47 - page 5, line 9 ---	
X	EICKEN VON T ET AL: "U-NET: A USER-LEVEL NETWORK INTERFACE FOR PARALLEL AND DISTRIBUTED COMPUTING" OPERATING SYSTEMS REVIEW (SIGOPS), vol. 29, no. 5, 1 December 1995 (1995-12-01), pages 40-53, XP000584816	1,4
Y		13,18
A		5,6, 8-10,16, 17
	page 43, left-hand column, line 56 - right-hand column, line 4 page 43, right-hand column, line 24 - line 32 page 44, left-hand column, line 13 - line 19 page 44, right-hand column, line 1 - line 33 page 45, left-hand column, line 31 - line 39 page 45, left-hand column, line 56 - line 59 page 45, right-hand column, line 25 - line 39 ---	
X	BURD D: "ZERO-COPY INTERFACING TO TCP/IP" DR. DOBB'S JOURNAL, vol. 20, no. 9, September 1995 (1995-09), pages 68, 70, 72, 74, 76, 78, 106, 108-110, XP000672215	1,4
A		2,3,5,6
	page 68, column 1, line 12 - line 18 page 68, column 3, line 10 - line 42 ---	
X	EP 0 790 564 A (TANDEM COMPUTERS INC) 20 August 1997 (1997-08-20)	1
A		2,4,5,17
	abstract column 3, line 10 - line 32 column 9, line 26 - column 11, line 29 ---	
	-/--	

2

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 98/24395

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	NEGISHI Y ET AL: "A PORTABLE COMMUNICATION SYSTEM FOR VIDEO-ON-DEMAND APPLICATIONS USING THE EXISTING INFRASTRUCTURE" PROCEEDINGS OF IEEE INFOCOM 1996. CONFERENCE ON COMPUTER COMMUNICATIONS, FIFTEENTH ANNUAL JOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES. NETWORKING THE NEXT GENERATION SAN FRANCISCO, MAR. 24 - 28, 1996, vol. 1, no. CONF. 15, 24 March 1996 (1996-03-24), pages 18-26, XP000622290 INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS	22,23
Y		20,21, 26-28
A	page 20, right-hand column, line 10 - page 21, left-hand column, line 7 page 25, right-hand column, line 6 - line 12 ---	1,2,7,29
X	O'BRYAN J ET AL: "XNS - X.25 COMMUNICATIONS GATEWAY" 21ST. CENTURY MILITARY COMMUNICATIONS - WHAT'S POSSIBLE ?, SAN DIEGO, OCT. 23 - 26, 1988, vol. 3, 23 October 1988 (1988-10-23), pages 1057-1061, XP000011148 INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS	22,23
A	page 1058, right-hand column, line 25 - line 33 page 1059, left-hand column, line 1 - line 22 page 1059, right-hand column, line 45 - line 50 ---	1,20,21, 29
Y	"TRANSMISSION CONTROL PROTOCOL/INTERNET PROTOCOL CHECKSUM IMPROVEMENT FOR AIX3.2 UNDER RISC SYSTEM/6000" IBM TECHNICAL DISCLOSURE BULLETIN, vol. 37, no. 2A, 1 February 1994 (1994-02-01), pages 253-256, XP000432638 ISSN: 0018-8689 page 255, line 11 - line 22 ---	26-28

-/--

2

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 98/24395

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 638 370 A (LEWIS GLENN ET AL) 10 June 1997 (1997-06-10) abstract column 3, line 41 - line 44 column 9, line 16 - line 25 column 14, line 49 - line 65 ---	26-28
P,X	US 5 701 316 A (MCBREEN JAMES R ET AL) 23 December 1997 (1997-12-23) abstract column 6, line 45 - line 52 column 7, line 1 - line 25 -----	26-28

2

International application No.
PCT/US 98/24395

INTERNATIONAL SEARCH REPORT

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. [ ] Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. [ ] Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. [ ] Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

SEE ADDITIONAL SHEET

1. [X] As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. [ ] As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. [ ] As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. [ ] No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

[ ] The additional search fees were accompanied by the applicant's protest.

[X] No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International Application No. PCT/US 98/24395

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. Claims: 1-19, 29-33

   Method to transfer network communicated data directly from network interface to application address space with only one physical copy of these data

2. Claims: 20-25

   Integrating separate protocol processing loops

3. Claims: 26-28

   Calculating protocol checksums one word at a time

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 98/24395

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
EP 0707266	A	17-04-1996	JP	9026929 A	28-01-1997
EP 0790564	A	20-08-1997	CA	2193343 A	21-06-1997
			JP	9325944 A	16-12-1997
US 5638370	A	10-06-1997	NONE		
US 5701316	A	23-12-1997	NONE		

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- [x] BLACK BORDERS
- [ ] IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- [ ] FADED TEXT OR DRAWING
- [ ] BLURRED OR ILLEGIBLE TEXT OR DRAWING
- [ ] SKEWED/SLANTED IMAGES
- [ ] COLOR OR BLACK AND WHITE PHOTOGRAPHS
- [ ] GRAY SCALE DOCUMENTS
- [ ] LINES OR MARKS ON ORIGINAL DOCUMENT
- [ ] REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- [ ] OTHER: ______________________

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.